# Text Data

Text is another example of a data type that is initially non-numerical and that must be processed before it can be fed into a machine learning algorithm. Let's have a look at some of the common tasks we might do as part of this processing.

## Normalization

One of the challenges that can come up in text analysis is that there are often multiple forms that mean the same thing. For example, the verb `to be` may show up as `is`, `am`, `are`, and so on. Or a document may contain alternative spellings of a word, such as `behavior` vs. `behaviour`. So one step that you will sometimes conduct in processing text is *normalization*.

> *Text **normalization** is the process of transforming a piece of text into a canonical (official) form.*

*Lemmatization* is an example of normalization. A **lemma** is the dictionary form of a word and **lemmatization** is the process of reducing multiple inflections to that single dictionary form. For example, we can apply this to the `is`, `am`, `are` example we mentioned above:

| Original word | Lemmatized word |
|---|---|
| is | be |
| are | be |
| am | be |

In many cases, you may also want to remove stop words. **Stop words** are high-frequency words that are unnecessary (or unwanted) during the analysis. For example, when you enter a query like `which cookbook has the best pancake recipe` into a search engine, the words `which` and `the` are far less relevant than `cookbook`, `pancake`, `recipe`. In this context, we might want to consider `which` and `the` to be stop words and remove them prior to analysis.

Here's another example:

| Original text | Normalized text |
|---|---|
| The quick fox. | [quick, fox] |
| The lazy dog. | [lazy, dog] |
| The rabid hare. | [rabid, hare] |

Here we have **tokenized** the text (i.e., split each string of text into a list of smaller parts or tokens), removed stop words (`the`), and standardized spelling (changing `lazy` to `lazy`).

QUESTION 1 OF 5

Here's another example:

| Original text | Normalized text |
|---|---|
| Mary had a little lamb. | [Mary, have, a, little, lamb] |
| Jack and Jill went up the hill. | [Jack, and, Jill, go, up, the, hill] |
| London bridge is falling down. | [London, bridge, be, fall, down] |

Looking at the normalized text, which of the following have been done?

- ✓ Tokenization
- ☐ Removal of stop words
- ✓ Lemmatization

SUBMIT

## Vectorization

After we have normalized the text, we can take the next step of *actually* encoding it in a numerical form. The goal here is to identify the particular features of the text that will be relevant to us for the particular task we want to perform—and then get those features extracted in a numerical form that is accessible to the machine learning algorithm. Typically this is done by text **vectorization**—that is, by turning a piece of text into a vector. Remember, a vector is simply an array of numbers—so there are many different ways that we can vectorize a word or a sentence, depending on how we want to use it. Common approaches include:

- Term Frequency-Inverse Document Frequency (TF-IDF) vectorization
- Word embedding, as done with Word2vec or Global Vectors (GloVe)

The details of these approaches are a bit outside the scope of this class, but let's take a closer look at TF-IDF as an example. The approach of TF-IDF is to give less importance to words that contain less information and are common in documents, such as "the" and "this"—and to give higher importance to words that contain relevant information and appear less frequently. Thus TF-IDF assigns weights to words that signify their relevance in the documents.

Here's what the word importance might look like if we apply it to our example

| quick | fox | lazy | dog | rabid | hare | the |
|---|---|---|---|---|---|---|
| 0.32 | 0.23 | 0.12 | 0.23 | 0.56 | 0.12 | 0.0 |

Here's what that might look like if we apply it to the normalized text:

| | quick | fox | lazy | dog | rabid | hare |
|---|---|---|---|---|---|---|
| [quick, fox] | **0.32** | **0.23** | 0.0 | 0.0 | 0.0 | 0.0 |
| [lazy, dog] | 0.0 | 0.0 | **0.12** | **0.23** | 0.0 | 0.0 |
| [rabid, hare] | 0.0 | 0.0 | 0.0 | 0.0 | **0.56** | **0.12** |

Noticed that "the" is removed since it has `0` importance here.

Each chunk of text gets a vector (represented here as a row in the table) that is the length of the total number of words that we are interested in (in this case, six words). If the normalized text does not have the word in question, then the value in that position is `0`, whereas if it does have the word in question, it gets assigned to the importance of the word.

QUESTION 2 OF 5

Let's pause to make sure this idea is clear. In the table above, what does the value `0.56` mean?

- ☐ It means that the word `fox` has some importance in `[quick, fox]`.
- ☐ It means that the word `rabid` has some importance in `[quick, fox]`.
- ☐ It means that the word `fox` has some importance in `[rabid, hare]`.
- ✓ It means that the word `rabid` has some importance in `[rabid, hare]`.

SUBMIT

QUESTION 3 OF 5

What vector will be used to represent "quick, lazy hare"

- ☐ (0.32, 0.23, 0.12, 0.0, 0.0, 0.0)
- ✓ (0.32, 0.0, 0.12, 0.0, 0.0, 0.12)
- ☐ (0.0, 0.0, 0.12, 0.0, 0.56, 0.12)
- ☐ (0.0, 0.0, 0.12, 0.23, 0.0, 0.12)

SUBMIT

## Feature Extraction

As we talked about earlier, the text in the example can be represented by vectors with length `6`. Since there are `6` words total.

[quick, fox] as `[0.32, 0.23, 0.0, 0.0, 0.0, 0.0]`

[lazy, dog] as `[0.0, 0.0, 0.12, 0.23, 0.0, 0.0]`

[rabid, hare] as `[0.0, 0.0, 0.0 , 0.0, 0.56, 0.12]`

We understand the text because each word has a meaning. But how do algorithms understand the text using the vectors, in other words, how do algorithms extract features from the vectors?

Vectors with length `n` can be visualized as a line in an `n` dimension space. For example, a vector `[1,1]` can be viewed as a line starting from (0, 0) and ending at `[1,1]`.


Any vector with the **same length** can be visualized in the **same space**. How close one vector is to another can be calculated as **vector distance**. If two vectors are close to each other, we can say the text represented by the two vectors have a similar meaning or have some connections. For example, if we add [lazy, fox] to our example:

| | quick | fox | lazy | dog | rabid | hare |
|---|---|---|---|---|---|---|
| [quick, fox] | **0.32** | **0.23** | 0.0 | 0.0 | 0.0 | 0.0 |
| [lazy, dog] | 0.0 | 0.0 | **0.12** | **0.23** | 0.0 | 0.0 |
| [rabid, hare] | 0.0 | 0.0 | 0.0 | 0.0 | **0.56** | **0.12** |
| [lazy, fox] | 0.0 | **0.23** | **0.12** | 0.0 | 0.0 | 0.0 |

Apparently, [lazy, fox] is more similar to [lazy, dog] than [rabid, hare], so the vector distance of [lazy, fox] and [lazy, dog] is smaller than that to [lazy, fox] and [rabid, hare].

QUESTION 4 OF 5

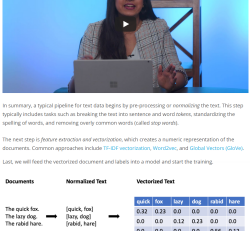
Imagine the words "monkey", "rabbit", "bird" and "raven" are represented by vectors with the same length. Based on the meanings of the words, which two words would we expect to have the smallest vector distance?

- ☐ "monkey" and "rabbit"
- ☐ "monkey" and "raven"
- ☐ "rabbit" and "bird"
- ✓ "raven" and "bird"

SUBMIT

## The Whole Pipeline

In this next video, we'll first review the above steps—normalization and vectorization—and then talk about how they fit into the larger goal of training a machine-learning model to analyze text data.



In summary, a typical pipeline for text data begins by pre-processing or normalizing the text. This step typically includes tasks such as breaking the text into sentence and word tokens, standardizing the spelling of words, and removing overly common words (called stop words).

*The next step is feature extraction and vectorization*, which creates a numeric representation of the documents. Common approaches include TF-IDF vectorization, Word2vec, and Global Vectors (GloVe).

Last, we will feed the vectorized document and labels into a model and start the training.

| Documents | | Normalized Text | | Vectorized Text | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | quick | fox | lazy | dog | rabid | hare |
| The quick fox. | ⟶ | [quick, fox] | ⟶ | 0.32 | 0.23 | 0.0 | 0.0 | 0.0 | 0.0 |
| The lazy dog. | | [lazy, dog] | | 0.0 | 0.0 | 0.12 | 0.23 | 0.0 | 0.0 |
| The rabid hare. | | [rabid, hare] | | 0.0 | 0.0 | 0.0 | 0.0 | 0.56 | 0.12 |

QUESTION 5 OF 5

What is the typical pipeline for a classification model using text data?

- ☐ vectorize text > normalize text > train model > deploy model
- ☐ train model > normalize text > vectorize text > deploy model
- ☐ vectorize text > normalize text > deploy model > train model
- ✓ normalize text > vectorize text > train model > deploy model

SUBMIT

NEXT